**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Бинарные деревья

|  |  |  |
| --- | --- | --- |
| Студент гр. 9304 |  | Борисовский В.Ю. |
| Преподаватель |  | Филатов Ар.Ю. |

Санкт-Петербург

2020

## Цель работы.

Ознакомиться с понятием бинарного дерева. Реализовать бинарное дерево для решения задания на языке программирования С++.

## Задание.

Вариант 6.

Задано бинарное дерево b типа ВТ с произвольным типом элементов. Используя очередь, напечатать все элементы дерева b по уровням: сначала - из корня дерева, затем (слева направо) - из узлов, сыновних по отношению к корню, затем (также слева направо) - из узлов, сыновних по отношению к этим узлам, и т. д.

## Выполнение работы.

## 1) Сперва я создал класс bin\_tree\_node, данный класс предназначен для хранения узла бинарного дерева, он имеет поля:

## std::shared\_ptr<bin\_tree\_node<T>> left - левое поддерево.

## std::shared\_ptr<bin\_tree\_node<T>> right - правое поддерево.

## T data - элемент шаблонного типа.

## 2) Затем был написан чекер строки, который проверяет ее на валидность. Чекер реализован в двух функциях - bool first\_checker(std::string &str, int &index), bool second\_checker(std::string &str). Первый чекер следит за тем, чтобы в случае когда выбрана опция «с» в конце строки в строке были только символы или же целые числа, если выбрана опция «i» (для определения опции также были написаны соответствующие чекеры). Первый чекер следит также за тем, что у корня дерева должно быть два наследника и не больше. Второй же чекер выполняет провреку на то, что после последней закрывающей скобки не следует лишних символов.

## 3) Затем я реализовал класс bin\_tree(). В конструкторе данного класса в начале проверяется чекером строка переданная для создания дерева, если чекер вернул true, значит все прошло успешно и для поля std::shared\_ptr<bin\_tree\_node<T>> head вызывается функция create\_bin\_tree(str, index), которая работает примерно как чекер, отличие в том , что в ней создаются узлы bin\_tree\_node и индексация происходит несколько иначе. Зная, что строка уже валидна эта функция легко проходится по данной строке и создает из нее дерево.

## 4) Дальше я выполнил условие задания своего варианта. Я реализовал обход бинарного дерева в ширину, который как раз и обеспечивает вывод всех элементов бинарного дерева по уровням с использованием очереди. Алгоритм данного обхода крайне прост: создается очередь, в нее кладется корень дерева. Затем пока очередь не станет пустой начинаем извлекать из нее первый элемент, и класть левого и правого наследника извлеченного элемента. Метод show\_tree().

## 5) После этого я реализовал функции LKP обхода, вставки и удаления элемента, требующиеся в условия лаб.работы.

## 6) В заключение была реализована функция main(), в ней из аргументов командой строки принимается строка, проверяется чекером опций «с» или же «i» затем создается объект класса bin\_tree и для него вызывается метод show\_tree(), иначе выводится сообщение о том, что строка неверна.

**Тестирование.**

Запуск программы начинается с ввода команды “make”, что приведёт к компиляции программы и созданию исполняемого файла lab3. Запуск программы производится командой ./lab3 и последующим вводом строки, содержащей логическое выражение. Тестирование производится с помощью

скрипта test\_skript.py. Запуск скрипта производится командой «python3 test\_skript.py» в директории lab3. Результаты тестирования представлены в приложении Б.

## Выводы.

Было изучено понятие бинарного дерева. Было реализовано бинарное дерево с помощью языка программирования С++, с использованием умных указателей std::shared\_ptr.

Была реализована программа, печатающая все элементы дерева b по уровням: сначала - из корня дерева, затем (слева направо) - из узлов, сыновних по отношению к корню, затем (также слева направо) - из узлов, сыновних по отношению к этим узлам, и т. д.

# Приложение А Исходный код программы

Название файла: main.cpp

#include <iostream>

#include <string>

#include <memory>

#include <queue>

#include <cstdlib>

//проверка на работу с char

bool char\_checker(std::string &str){

if (str[str.length() - 1] == 'c'){

return true;

} else {

return false;

}

}

//проверка на работу с int

bool int\_checker(std::string &str){

if (str[str.length() - 1] == 'i'){

return true;

} else {

return false;

}

}

//валидация строка до последней закрывающей скобки

bool first\_checker(std::string &str, int &index){

if (str[index] != '('){

return false;

}

index++;

if (char\_checker(str)) {

if (str[index] != '(' && str[index] != ')') {

index++;

} else if (str[index] == ')') {

return true;

} else if (str[index] == '(') {

return false;

}

if (str[index] == ')') {

return true;

}

} else if (int\_checker(str)){

if (str[index] != '(' && str[index] != ')') {

std::string check\_num = "";

while (str[index] != '(' && str[index] != ')') {

check\_num += str[index];

index++;

}

if (!check\_num.empty()) {

char \*endptr;

const char \*c\_string = check\_num.c\_str();

strtol(c\_string, &endptr, 10);

if (\*endptr) {

return false;

}

} else {

return false;

}

} else if (str[index] == ')') {

return true;

} else if (str[index] == '(') {

return false;

}

if (str[index] == ')') {

return true;

}

}

if (!first\_checker(str, index)){

return false;

}

index++;

if (!first\_checker(str, index)){

return false;

}

index++;

if (str[index] == ')'){

return true;

} else {

return false;

}

}

//проверка на то, что строка кончается валидно

bool second\_checker(std::string &str){

int index = 0;

if(first\_checker(str, index) && index + 2 == str.length()){

return true;

} else {

return false;

}

}

//структура узла бинарного дерева

template <typename T>

class bin\_tree\_node{

public:

std::shared\_ptr<bin\_tree\_node<T>> left;

std::shared\_ptr<bin\_tree\_node<T>> right;

T data;

};

//бинарное дерево

template<typename T>

class bin\_tree{

public:

std::shared\_ptr<bin\_tree\_node<T>> head;

//реализация конструктора (валидация строки)

bin\_tree(std::string &str){

int index = 0;

bool check = second\_checker(str);

if (check){

head = create\_bin\_tree(str, index);

} else {

std::cout << "wrong string\n";

head = nullptr;

}

}

~bin\_tree() = default;

//создание бинарного дерева

std::shared\_ptr<bin\_tree\_node<T>> create\_bin\_tree(std::string &str, int &index){

std::shared\_ptr<bin\_tree\_node<T>> node = std::make\_shared<bin\_tree\_node<T>>();

index++;

if (str[index] == ')'){

index++;

return nullptr;

} else {

if (int\_checker(str)) {

std::string check\_num = "";

while (str[index] != '(' && str[index] != ')') {

check\_num += str[index];

index++;

}

node->data = strtol(check\_num.c\_str(), NULL, 10);

} else if (char\_checker(str)){

node -> data = str[index];

index++;

}

}

if (str[index] == ')'){

node -> left = nullptr;

node -> right = nullptr;

index++;

return node;

} else {

node -> left = create\_bin\_tree(str, index);

node -> right = create\_bin\_tree(str, index);

index ++;

return node;

}

}

//выполнения задания лабороторной (посредством обхода в ширину)

void show\_tree(){

std::cout << "success: ";

std::queue<std::shared\_ptr<bin\_tree\_node<T>>> queue;

queue.push(head);

while (!queue.empty()) {

std::shared\_ptr<bin\_tree\_node<T>> tmp = queue.front();

queue.pop();

std::cout << tmp->data << " ";

if (tmp -> left) {

queue.push(tmp->left);

}

if (tmp -> right) {

queue.push(tmp->right);

}

}

}

//LKP обход

void LKP(std::shared\_ptr<bin\_tree\_node<T>> hd){

if (hd -> left){

LKP(hd -> left);

}

std::cout << hd -> data << " ";

if (hd -> right){

LKP(hd -> right);

}

}

//вставка элемента

void insert\_elem(std::shared\_ptr<bin\_tree\_node<T>> hd, T elem){

while(hd->left) {

hd = hd->left;

}

hd->left = std::make\_shared<bin\_tree\_node<T>>();

hd->left->left = nullptr;

hd->left->right = nullptr;

hd->left->data = elem;

}

//удаление элемента

void delete\_elem(std::shared\_ptr<bin\_tree\_node<T>> hd, T elem){

if(hd->left != nullptr && hd->left -> data == elem){

hd->left = nullptr;

}

if(hd -> right != nullptr && hd -> right -> data == elem){

hd -> right = nullptr;

}

if(hd -> left != nullptr)

delete\_elem(hd -> left, elem);

if(hd -> right != nullptr)

delete\_elem(hd -> right, elem);

}

};

int main(int argc, char\* argv[]){

if(argc == 1){

std::cout << "Wrong expression\n";

return 0;

}

std::string str(argv[1]);

if (char\_checker(str)){

bin\_tree<char> bt(str);

if (bt.head){

bt.show\_tree();

}

} else if (int\_checker(str)){

bin\_tree<int> bt(str);

if (bt.head){

bt.show\_tree();

}

} else {

std::cout << "wrong string\n";

}

}

# Приложение Б Тестирование

Результаты тестирования представлены в таблице Б.1

Таблица Б.1 — Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Результат проверки |
| 1. | (a(b)(c))c | success: a b c | success |
| 2. | (v(c(n)(x))())c | success: v c n x | success |
| 3. | (10(100(30(40)(70))())(12))i | success: 10 100 12 30 40 70 | success |
| 4. | (a(b)(c)))c | wrong string | wrong string |
| 5. | (1y)(09)9i | wrong string | wrong string |
| 6. | (a(b)(c))i | wrong string | wrong string |